The Complexity of the Path-following Solutions of Two-dimensional Sperner/Brouwer Functions

Paul W. Goldberg*

Department of Computer Science
University of Oxford
paul.goldberg@cs.ox.ac.uk

Abstract. There are a number of results saying that for certain "path-following" algorithms that solve PPAD-complete problems, the solution obtained by the algorithm is PSPACE-complete to compute. We conjecture that these results are special cases of a much more general principle, that all such algorithms compute PSPACE-complete solutions. Such a general result might shed new light on the complexity class PPAD.
In this paper we present a new PSPACE-completeness result for an interesting challenge instance for this conjecture. Chen and Deng [1] showed that it is PPAD-complete to find a trichromatic triangle in a concisely-represented Sperner triangulation. The proof of Sperner's lemma — that such a solution always exists — identifies one solution in particular, that is found via a natural "path-following" approach. Here we show that it is PSPACE-complete to compute this specific solution, together with a similar result for the computation of the path-following solution of two-dimensional discrete Brouwer functions.

1 Preliminaries

We begin with the definitions needed for the result presented here, then we mention some related work and motivation. We consider search problems where any instance I has a set S_I of associated solutions represented by bit strings of length bounded by some polynomial in the length of I; the challenge is to find an element of S_I. A search problem is in NP if there is a polynomial-time algorithm that can check, given I and a bit string s, whether s belongs to S_I. A search problem is *total* if for all I, S_I is non-empty.

Definition 1.1. *Let X and Y be two* NP *search problems. A* reduction (f, g) from X to Y *consists of polynomial-time computable functions f and g, where f maps instances of X to instances of Y, and g maps solutions of Y to solutions of X, such that $g(f(I))$ is a solution of I.*

Papadimitriou [9] introduced the complexity class PPAD, defined in terms of the problem End of line (Definition 1.2). A search problem X belongs to PPAD if there exists a polynomial-time reduction from X to End of line, and X is PPAD-complete if X belongs to PPAD, and End of line is reducible to X. End of line is an NP total search problem that appears to be hard.

Definition 1.2. *An instance of* End of line *consists of two directed boolean circuits P, S, each having n input nodes and n output nodes, and each of size polynomial in n. Consider*

* Supported by EPSRC under grant EP/K01000X/1

an associated directed graph $G(P, S)$ on the 2^n bit strings, obtained by including arc (u, v) whenever $S(u) = v$ and $P(v) = u$. The exception to this rule is that the all-zeroes bit string $\mathbf{0}$ *is deemed to have no incoming edge.*

The problem is to search for a bit string, other than $\mathbf{0}$*, that belongs to exactly one arc (either as a source or a sink).*

Since, by construction, vertices of $G(P, S)$ have at most one incoming arc and at most one outgoing arc, there is an "obvious" exponential-time algorithm that finds a solution to End of line as follows. Starting at the vertex $\mathbf{0}$, compute $S(\mathbf{0})$, $S(S(\mathbf{0}))$, $S(S(S(\mathbf{0})))$, $\cdots$, until a vertex is found that has no outgoing arc. We call this the path-following approach. The other natural exponential-time algorithm is lexicographic search. This paper aims to develop a deeper understanding of the properties of path-following solutions, and how they contrast with lexicographic search solutions. Notice that:

- End of line is a *total* search problem: any instance (P, S) has a solution, and the proof of existence identified the path-following solution.
- Moreover, a solution to an instance (P, S) is allowed to be *any* degree-1 vertex (other than $\mathbf{0}$); as a consequence, End of line is a search problem that belongs to NP (it is easy to use P and S to check that a given solution is valid).
- Although the unique path-following solution can be easily verified as a valid solution, its status as the path-following solution cannot be efficiently verified.

We continue with the definitions of the PPAD-complete problems referred to in the title, using essentially the same definitions given in [1] (although we prefer to describe them in terms of boolean circuits rather than Turing machines); we will see that these problems also have natural path-following algorithms.

Definition 1.3. *An instance of* 2D-discrete-Brouwer *consists of a directed Boolean circuit C having $2n$ input nodes and 2 output nodes; C is of size polynomial in n. We assume that the values taken by the output nodes represent three colour values $\{0, 1, 2\}$ (for example, by taking* 00 *and* 01 *to represent* 0*,* 10 *to represent 1, and* 11 *to represent* 2*). C represents a colouring of the integer grid points (x, y), $0 \le x, y < 2^n$ as follows. The input nodes represent the coordinates of point (x, y), and the outputs represent its colour. We impose the following* boundary condition*, that all points $(0, y)$ must have colour 1, points $(x, 0)$ other than the origin have colour 2, and other points of the form $(x, 2^n - 1)$ or $(2^n - 1, y)$ have colour 0.*

The problem is to search for a unit square whose vertices contain points with all 3 colours.

2D-discrete-Brouwer has an algorithm (given in detail in Definition 1.6) that can be described informally as follows. Notice that a discrete Brouwer function divides the grid into a number of coloured regions, where regions are connected components of points having the same colour, that can be reached from each other via unit-length edges, or diagonal edges connecting (x, y) with $(x+1, y-1)$. We seek a point where all three colours are close together. The boundary condition identifies a known point where regions having colours 1 and 2 are adjacent. Suppose that we trace the boundary of these two regions; then we either encounter the third colour, or we exit the rectangle. However, the boundary condition says that there

is only one boundary point where colours 1 and 2 are adjacent, so in fact we encounter the third colour.

Figure 1 (left-hand side) shows an example of a simple discrete Brouwer function, together with the path followed by the above algorithm.

Definition 1.4. *An instance of* Sperner *consists of a colouring of the set of non-negative integer grid points (x, y) satisfying $x + y < 2^n$. The colouring is specified using a circuit C of size polynomial in n, taking as input the coordinates (x, y) and outputting a colour in $\{0, 1, 2\}$, and satisfying the boundary conditions that no point with $x = 0$ gets colour 0, no point with $y = 0$ gets colour 1, and no point with $x + y = 2^n - 1$ gets colour 2. Consider the triangulation of the grid obtained by connecting pairs of points in this domain that are distance 1 apart, also all pairs of points $\{(x, y), (x - 1, y + 1)\}$.*

The problem is to search for a "trichromatic triangle", that is, one of these triangles having 3 distinct colours at its vertices.

Next, we review a straightforward reduction from 2D-discrete-Brouwer to Sperner; the reduction serves to identify some of the notation we use. The reduction is essentially a slight simplification of one presented in [1]; we will use it later to show that the path-following solution of Sperner (obtained by the algorithm of Definition 1.7) is also PSPACE-complete to compute.

Reducing 2D-discrete-Brouwer *to* Sperner. Let I be an instance of 2D-discrete-Brouwer with size parameter value n. f constructs an instance I' of Sperner with size parameter value $n+2$, where every point (x, y) in I has 4 corresponding points in I' coloured as follows (Figure 1 shows an example):

- In I', $(2x, 2y)$, $(2x + 1, 2y)$, and $(2x, 2y + 1)$ receive the same colour as point (x, y) in I. Point $(2x - 1, 2y - 1)$ receives the colour of (x, y) in I, for all $x, y > 0$.
- Points on the boundary obey the boundary condition of Sperner as follows: the origin is coloured 2; all points $(0, y)$ with $y > 0$ are coloured 1; all points $(x, 0)$ with $x < 2^{n+1}$ are coloured 2; with $x \geq 2^{n+1}$ are coloured 0.
- Remaining points are coloured 0.

g takes the triangle t that constitutes a solution of I', and outputs the square in I that, if all its coordinates were doubled, contains t.

Definition 1.5. *A* path-following algorithm $\mathcal{A}$ *for a* PPAD*-complete problem X is defined in terms of a reduction (f, g) from X to* End of line*. Given an instance I of X, $\mathcal{A}$ computes $f(I)$, then it follows the path in $f(I)$ beginning at the all-zeroes string $\mathbf{0}$, obtaining a sink x in the* End of line *graph defined by $f(I)$. The algorithm outputs $g(x)$.*

Path-following algorithms take time exponential in n, but polynomial space. Lexicographic search algorithms are similar in this respect, but have a fundamentally different nature. The solutions to all known path-following algorithms are PSPACE-complete to compute, while the solutions to lexicographic search problems have lower computational complexity, essentially in the complexity class OptP (Krentel [7]) characterising the complexity of finding the lexicographically first satisfying assignment of an NP search problem. Notice that an incorrect solution s has a short certificate, consisting of a solution s' that precedes s lexicographically.

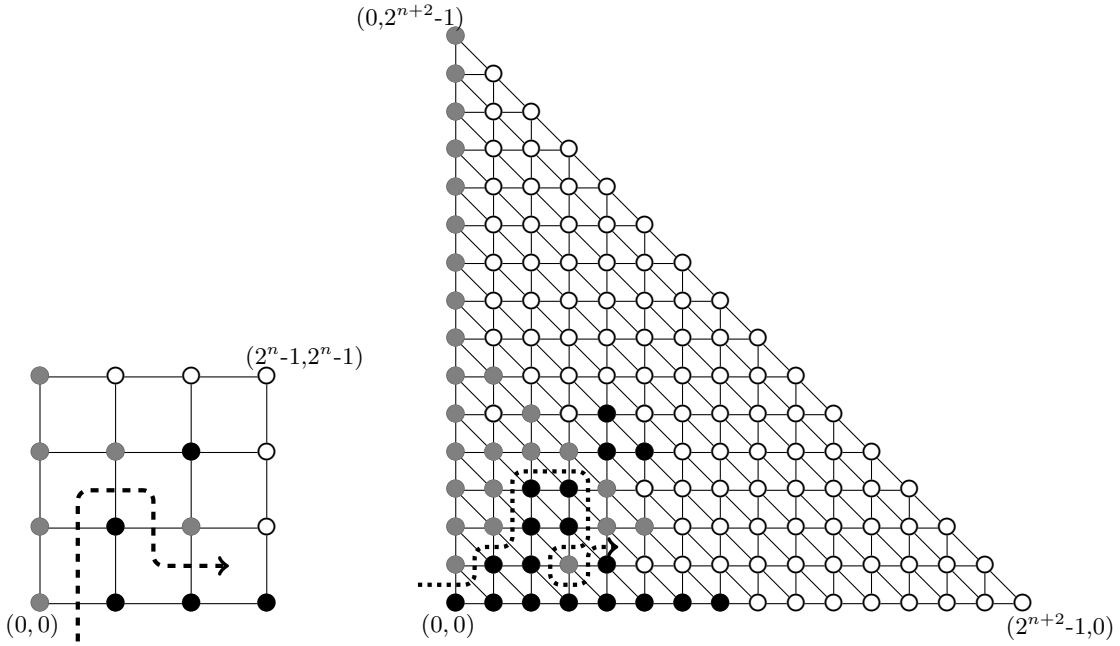

Fig. 1. Example of the reduction from 2D-DISCRETE-BROUWER to SPERNER. On the left is a discrete Brouwer function and on the right is the derived Sperner triangulation. A white point denotes 0, grey denotes 1, black denotes 2. Dashed arrows show the paths followed by the "natural" path-following algorithms.

We next define path-following algorithms for the two problems under consideration.

Definition 1.6. *The* natural path-following algorithm *for* 2D-DISCRETE-BROUWER *is defined as follows.*

1. *Let S be the bottom left-hand square of the grid; by construction S contains vertices coloured 1 and 2.*
2. *If S contains vertices coloured 0, 1, and 2, halt and output S*
3. *Else, let S' be a square adjacent to S that shares an edge with vertices v_1 and v_2 coloured 1 and 2, such that v_1 is to the left of v_2 when viewed from S. If 2 such squares exist, select the one that results in a right turn from the previous direction moved.*
4. *Set $S = S'$ and return to Step 2.*

To see that this is a path-following algorithm according to Definition 1.5, note that the constructed END OF LINE graph has a node for for every square in the grid, with an additional node for any square having vertices coloured 1,2,1,2 in clockwise order, and arcs that connect squares in the way indicated in the algorithm. Figure 1 (LHS) shows the path leading to the solution at the bottom right square.

Definition 1.7. *The* natural path-following algorithm *for* SPERNER *is defined as follows.*

1. *Given an instance I of* SPERNER*, construct an extended triangulation T by adding a point p to the left, coloured 1, and edges between p and all grid points $(0, y)$.*

2. *Let G be a graph where each node corresponds with a triangle of T.*
3. *If triangle t' can be reached from triangle t by crossing an edge with a vertex coloured 1 on the left and 2 on the right, there is an arc in G between the corresponding nodes of G.*
4. *By construction, the triangle with vertices $\{p, (0,0), (0,1)\}$ has no incoming edge and one outgoing edge; follow the path in G that begins at this triangle.*

This algorithm constitutes the standard proof that SPERNER is indeed a total search problem. Figure 1 (RHS) shows the path leading to the unique solution found by this algorithm. p and its edges are not shown; in the example the extra triangles do not occur in the path.

Theorem 1.1. *It is* PSPACE*-complete to compute the output of the natural path-following algorithm, applied to instances of* 2D-DISCRETE-BROUWER*.*

Theorem 1.1 is proved in the next section. Before that, we note the following straightforward corollary.

Corollary 1.1. *It is* PSPACE*-complete to compute the output of the natural path-following algorithm, applied to instances of* SPERNER*.*

Proof. Consider the reduction we described, from 2D-DISCRETE-BROUWER to SPERNER. Let I be an instance of 2D-DISCRETE-BROUWER and I' the corresponding instance of SPERNER. If s is a triangle in I' that lies in the square $(2x, 2y), (2x+2, 2y), (2x+2, 2y+2), (2x, 2y+2)$, let $g(s)$ be the square $(x, y), (x+1, y), (x+1, y+1), (x, y+1)$. It can be checked that:

- if s is a solution of I' then $g(s)$ is a solution of I.
- If s and s' are consecutive under the natural path-following algorithm for SPERNER, then either $g(s) = g(s')$, or $g(s)$ and $g(s')$ are consecutive under the natural path-following algorithm for 2D-DISCRETE-BROUWER. Conversely, if $g(s)$ and $g(s')$ are equal or consecutive, then there is a short path from s to s'.

This means that the path-following algorithm as given for SPERNER mimics the path-following algorithm as given for 2D-DISCRETE-BROUWER and if s is the path-following solution of I' then $g(s)$ is the path-following solution of I. □

Motivation, and some related work: It is known from [3] that it is PSPACE-complete to find the solution of instances (P, S) of END OF LINE consisting of the unique sink that is connected to the given source $\mathbf{0}$ in $G(P, S)$. That sink is the "path-following solution" of (P, S). (In [5] this challenge is called EOTPL for "End of this particular line".) A well-known result of algorithmic game theory is the PPAD-completeness of computing a Nash equilibrium of a given bimatrix game [2,4]. Now, there are certain path-following algorithms that compute Nash equilibria, and these algorithms are of interest as a theory of equilibrium selection [6]. The best-known path-following algorithm for Nash equilibrium computation is the Lemke-Howson algorithm [8], and it has been shown to be PSPACE-complete to compute the outputs of Lemke-Howson and related algorithms [5].

This paper is partly motivated by the "paradox" that the Lemke-Howson algorithm is efficient in practice, although the computational complexity of its solutions is much higher than

unrestricted solutions, or even selected solutions such as the lexicographically-first equilibrium. Relevant to this is the possibility of a general principle, saying that given any PPAD-complete problem X, and a path-following algorithm $\mathcal{A}$ for X, that the outputs of $\mathcal{A}$ are PSPACE-complete to compute.

The problem under consideration, 2D-DISCRETE-BROUWER, represented a challenge instance for this conjecture. To see why, it is helpful to contrast it with its three-dimensional analog, 3D-DISCRETE BROUWER, originally shown to be PPAD-complete by Papadimitriou [9] in 1994. Let (f, g) be the reduction of [9] from END OF LINE to 3D-DISCRETE BROUWER, and let (f', g') be the reduction from 3D-DISCRETE BROUWER to END OF LINE that corresponds with the natural path-following algorithm from 3D-DISCRETE BROUWER (where we begin at the origin, and follow a sequence of cubes that have colours 1,2, and 3, until colour 0 is encountered). It can be checked that an END OF LINE graph $G(P, S)$ is topologically similar to the graph $G(f'(f(P, S)))$ in the sense that given any nodes v_1, v_2 in $G(P, S)$ at opposite ends of a directed path, there exist nodes v_1', v_2' in $G(f'(f(P, S)))$ at opposite ends of a directed path, such that $g(g'(v_1')) = v_1$ and $g(g'(v_2')) = v_2$. Moreover, $g(g'(\mathbf{0})) = \mathbf{0}$, the directions of these paths remains the same, and the correspondence between these pairs of endpoints is 1-1. For our purposes, the main point to note is that if v is the sink of $G(f'(f(P, S)))$ connected to $\mathbf{0}$ then $g(g'(v))$ is the sink connected to $\mathbf{0}$ in $G(P, S)$. It follows that it's PSPACE-complete to compute the path-following solution to 3D-DISCRETE BROUWER, since the path-following solution of $G(P, S)$ is PSPACE-complete to compute.

Now consider the 2009 PPAD-completeness proof of 2D-DISCRETE-BROUWER in [1]. Applying the above notation to their reduction, when we compare $G(P, S)$ with $G(f'(f(P, S)))$, the only thing that is preserved is the number of endpoints of paths. In other respects, the topology is changed drastically, and in general if v the vertex connected to $\mathbf{0}$ in $G(f'(f(P, S)))$, it need not the case that $g(g'(v))$ is connected to $\mathbf{0}$ in $G(P, S)$. This problem is incurable, since when we attempt to design a scheme for embedding a large number of edges in the plane, there will typically be crossing-points, and these crossing-points are handled by [1] using a gadget that changes the topology of the graph.

In our PSPACE-completeness result for 2D-DISCRETE-BROUWER, it turns out to be convenient to reduce from QBF; this is in contrast with the PSPACE-completeness results discussed above, which reduce from the problem of computing the configuration reached by a space bounded Turing machine after exponentially-many steps.

2 Proof of Theorem 1.1

We define a gadget that is used throughout the proof.

Definition 2.1. *A* wire *consists of a sequence of grid points $p_1, p_2, \ldots$ coloured 1, together with a sequence of grid points $p_1', p_2', \ldots$ coloured 2, with the following properties. The distances between any p_i and p_{i+1}, and between any p_i' and p_{i+1}', is 1. If we move from p_i to p_{i+1}, at least one point on the right is some point p_j', and least one point on the left is coloured 0; similarly, if we move from p_i' to p_{i+1}', at least one point on the left is some p_j and at least one point on the right is coloured 0.*

Definition 2.1 is designed to specify a (directed) path in the plane that must be followed by the natural path-following algorithm for 2D-DISCRETE-BROUWER, assuming that the path being followed separates regions coloured 1 and 2. Figure 2 shows an example of a wire.

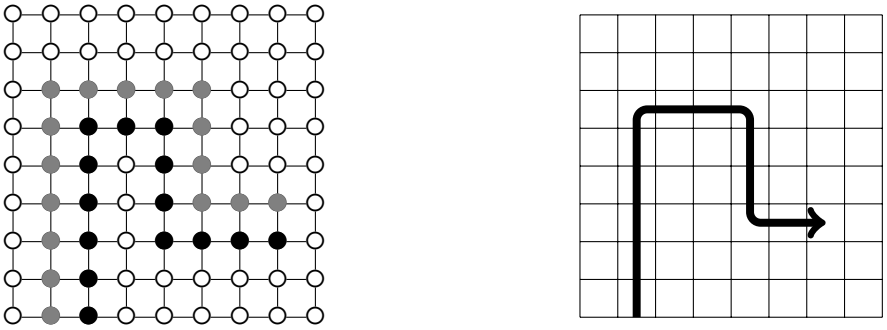

Fig. 2. Wire example (left-hand side), and a simplified depiction (right-hand side) used in subsequent diagrams, showing the directed path taken by the natural path-following algorithm.

Suppose that an instance of 2D-DISCRETE-BROUWER is composed entirely of wires. Notice that an endpoint of a wire —other than the one at the origin— is located at a solution of 2D-DISCRETE-BROUWER, and any solution of 2D-DISCRETE-BROUWER is located at the endpoints of wires.

We reduce from QBF, the problem of checking whether a given quantified boolean formula with no free variables evaluates to true. Let

$$\Phi = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \phi$$

where ϕ is a propositional formula over variables $x_1 \dots x_n$ and each Q_i is a quantifier. Define an i-subformula of Φ to be a QBF derived from Φ by fixing the values of variables $x_1, \dots, x_i$. Thus there are 2^i i-subformulae of Φ. A 0-subformula of Φ is just Φ itself. If $\mathbf{x}$ is a bit string of length i let $\Phi_{\mathbf{x}}$ be the i-subformula obtained by setting $x_1, \dots, x_i$ to $\mathbf{x}$.

Given Φ, we construct an instance of 2D-DISCRETE-BROUWER given by a circuit C_Φ of size polynomial in n, where C_Φ colours points in the integer grid having coordinates that may be exponential in n. Each i-subformula $\Phi_{\mathbf{x}}$ has an associated structure $\mathcal{S}(\Phi_{\mathbf{x}})$ in the planar grid. Each such structure has an incoming wire on its left; on its right it has two outgoing wires, with an incoming wire between them. The intention is that the structure will have the following property:

Property 1: if $\Phi_{\mathbf{x}}$ evaluates to TRUE, then the incoming wire on the left of $\mathcal{S}(\Phi_{\mathbf{x}})$ will link to the bottom outgoing wire on the right, otherwise it will link to the top outgoing wire. The other two wires will link together.

Details of the construction of $\mathcal{S}(\Phi_{\mathbf{x}})$: The location of $\mathcal{S}(\Phi_{\mathbf{x}})$ is as follows:

– $\mathcal{S}(\Phi_{\mathbf{x}})$ has a bounding box with height and width proportional to $2^{n-|\mathbf{x}|}$. The coordinates of the bounding box are easy to compute from Φ and $\mathbf{x}$.
– The bounding box of $\mathcal{S}(\Phi_{\mathbf{x}})$ contains the bounding boxes of $\mathcal{S}(\Phi_{\mathbf{x}0})$ and $\mathcal{S}(\Phi_{\mathbf{x}1})$, which are arranged side-by-side (with a small gap between them) with their centres having the same y-coordinate. (See Figure 6).
– For $|\mathbf{x}| = n$, structures $\mathcal{S}(\Phi_{\mathbf{x}})$ are arranged in a horizontal line in increasing order of $\mathbf{x}$.

Each bounding box has on its left-hand side, an "incoming terminal", a site where a wire inside the bounding box ends at the left-hand side, and is directed towards the interior of the box. On the right-hand side there are two outgoing terminals (ends of wires directed outwards) and another incoming terminal between them (See Figures 3, 4, 5). The coordinates of these terminals are, of course, easy to compute from Φ and $\mathbf{x}$.

For $|\mathbf{x}| = n$, if $\mathbf{x}$ satisfies ϕ, $\mathcal{S}(\Phi_{\mathbf{x}})$ is as depicted on the LHS of Figure 3, else as shown on the RHS. Of course, satisfaction of ϕ can be checked by a poly-sized boolean circuit that takes as input the coordinates of the location of $\mathcal{S}(\Phi_{\mathbf{x}})$ and places the correct structure. Note that Property 1 is satisfied by this design.

For $|\mathbf{x}| < n$, each i-subformula $\Phi_{\mathbf{x}}$ has structure $\mathcal{S}(\Phi_{\mathbf{x}})$ obtained by connecting up the structures $\mathcal{S}(\Phi_{\mathbf{x}0})$ and $\mathcal{S}(\Phi_{\mathbf{x}1})$ as described below. By construction these two sub-structures are consecutive and adjacent: $\mathcal{S}(\Phi_{\mathbf{x}0})$ just to the left of $\mathcal{S}(\Phi_{\mathbf{x}1})$. Assume inductively that Property 1 is satisfied by these structures. We connect up their incoming and outgoing wires as shown in Figure 4 if $Q_{i+1} = \forall$ and as shown in Figure 5 if $Q_{i+1} = \exists$.

The outermost structure $\mathcal{S}(\Phi)$ is connected by a wire starting from the known source at the bottom left-hand corner of the domain, as shown in Figure 6.

It is useful in the subsequent proof to highlight the following observation:

Observation 1 *Within any $\mathcal{S}(\Phi_{\mathbf{x}})$, the topological structure of the wires is that each incoming terminal leads to an outgoing terminal, and there are no internal ends of wires.*

Proof that the connections encode the truth value of Φ: We claim that Φ evaluates to TRUE if and only if the wire that begins at the origin ends at the lower outgoing terminal of $\mathcal{S}(\Phi)$, otherwise the wire ends at the upper outgoing terminal (labelled NO in Figure 6).

This is proved to hold for all structures $\mathcal{S}(\Phi_{\mathbf{x}})$ by backwards induction on the length of $\mathbf{x}$. It can be seen to hold for $|\mathbf{x}| = n$, from Figure 3. We show that it holds for $|\mathbf{x}| = i < n$, assuming that it holds for all $\mathbf{x}$ with $|\mathbf{x}| > i$.

Suppose first that $\Phi_{\mathbf{x}}$ is a universally quantified subformula, thus $\Phi_{\mathbf{x}} = \forall x_{i+1} \cdots$. Thus $\Phi_{\mathbf{x}}$ is satisfied if $\Phi_{\mathbf{x}0}$ and $\Phi_{\mathbf{x}1}$ are both satisfied. We connect up $\mathcal{S}(\Phi_{\mathbf{x}0})$ and $\mathcal{S}(\Phi_{\mathbf{x}1})$ as shown in Figure 4. Suppose $\Phi_{\mathbf{x}0}$ and $\Phi_{\mathbf{x}1}$ are indeed both satisfied. By the inductive hypothesis, the incoming wire on the left connects with the outgoing wire from $\mathcal{S}(\Phi_{\mathbf{x}0})$ labelled YES, which then connects via $\mathcal{S}(\Phi_{\mathbf{x}1})$ to the RHS outgoing YES wire. If $\Phi_{\mathbf{x}0}$ is not satisfied, the incoming wire connects to the outgoing NO wire of $\mathcal{S}(\Phi_{\mathbf{x}0})$, and thence directly to the RHS NO wire. If $\Phi_{\mathbf{x}0}$ is satisfied, but not $\Phi_{\mathbf{x}1}$, then we reach the outgoing NO wire of $\mathcal{S}(\Phi_{\mathbf{x}1})$, which feeds back to the RHS incoming wire of $\mathcal{S}(\Phi_{\mathbf{x}0})$, and by Observation 1, the only way out of $\mathcal{S}(\Phi_{\mathbf{x}0})$ is via its outgoing NO wire (since $\Phi_{\mathbf{x}0}$ is satisfied, the outgoing YES wire connects to the incoming LHS wire), which takes us to the outgoing NO wire of $\mathcal{S}(\Phi_{\mathbf{x}})$.

The argument for the existential quantifier is similar, with respect to Figure 5.

Fig. 3. Structures associated with length-n bit vectors $\mathbf{x}$. The dashed lines show the bounding boxes. The black wires are internal to the structures, while the wires shown in grey are part of larger structures $\mathcal{S}(\Phi_{\mathbf{y}})$ for $\mathbf{y}$ a prefix of $\mathbf{x}$.

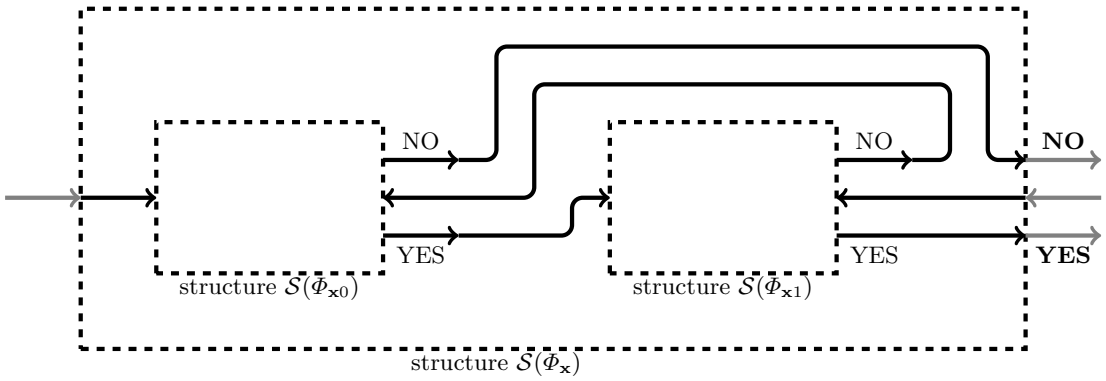

Fig. 4. combine 2 universally quantified sub-formulae: $\mathcal{S}(\Phi_{\mathbf{x}})$ connects the incoming and outgoing terminals of $\mathcal{S}(\Phi_{\mathbf{x}0})$ and $\mathcal{S}(\Phi_{\mathbf{x}1})$ using the wires shown in black; wires shown in grey are part of a larger structure.

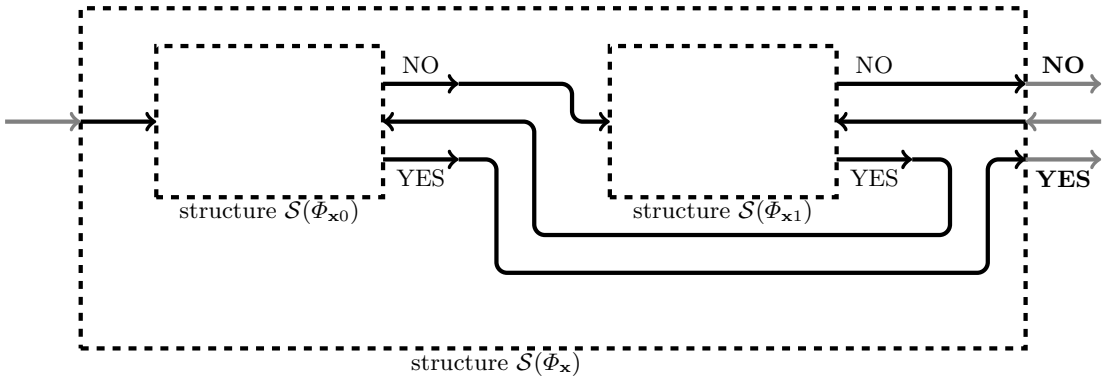

Fig. 5. combine 2 existentially quantified sub-formulae: $\mathcal{S}(\Phi_{\mathbf{x}})$ connects the incoming and outgoing terminals of $\mathcal{S}(\Phi_{\mathbf{x}0})$ and $\mathcal{S}(\Phi_{\mathbf{x}1})$ using the wires shown in black; wires shown in grey are part of a larger structure.

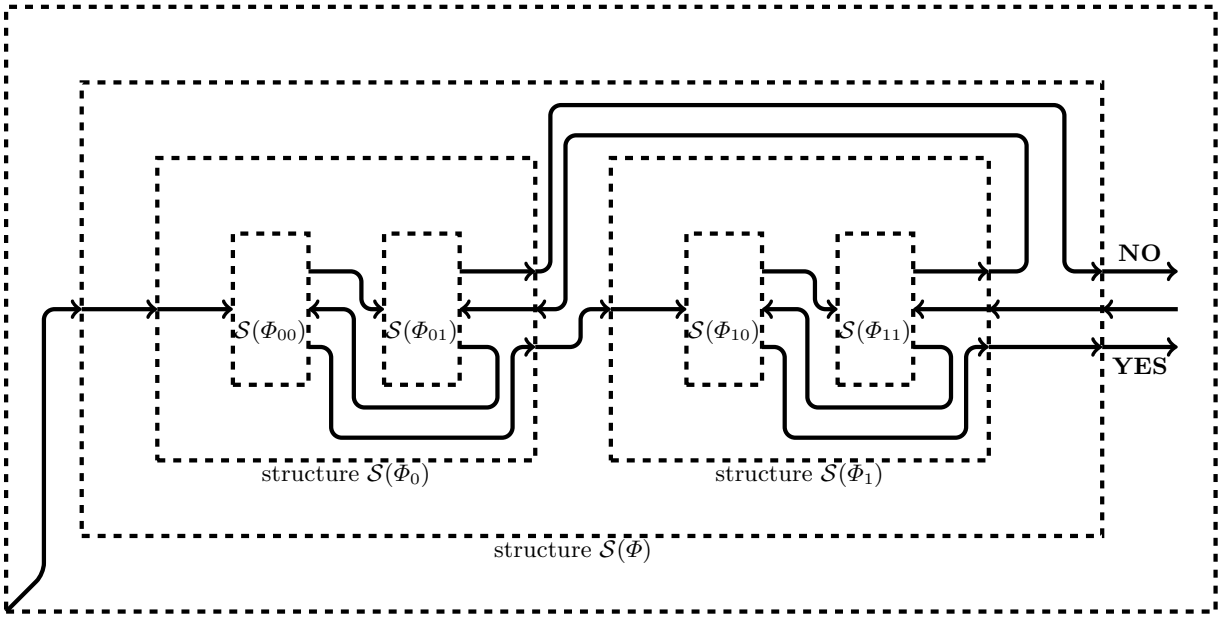

Fig. 6. Encoding a formula Φ of the form $\forall x_1 \exists x_2 \cdots$ (not to scale). The outermost dashed rectangle represents the entire domain of the corresponding instance of 2D-DISCRETE-BROUWER. Recall that the bottom edge of the domain is coloured 2, the left edge is coloured 1, and the other edges are coloured 0. Hence the bottom-left square is known to contain colours 1 and 2. The arrows show the wires connecting the (nested) structures that correspond to Φ itself, Φ with x_1 set to 0 or 1, and Φ with the four alternative settings of x_1 and x_2.
The wire that begins at the origin will lead to the solution labelled YES if Φ evaluates to TRUE, and NO if Φ evaluates to FALSE.

3 Conclusion and Further Work

With our current state of knowledge, it may be that for every PPAD-complete problem X, and every path-following algorithm $\mathcal{A}$ for X, it is PSPACE-complete to compute the output of $\mathcal{A}$ on instances of X. The reason to address this question is that it may shed light on the nature of PPAD-completeness, perhaps helping to relate PPAD to other complexity classes. We cannot go further and claim that it may be PSPACE-complete to compute the output of any exponential-time algorithm, since the output of lexicographic search has complexity below PSPACE.

A straightforward corollary of our main result is that if we are allowed to start at any point on the boundary where two colours meet, and trace the path that begins there until the third colour is reached, that the problem remains PSPACE-complete. This is reminiscent of the result of [5] that the problem of computing a Lemke-Howson solution of a game remains PSPACE-complete even if the algorithm is free to choose the initially dropped label. We conjecture that these are instances of a more general principle.

References

1. Chen, X., Deng, X.: On the Complexity of 2D Discrete Fixed Point Problem. Theoretical Computer Science 410(44), 4448–4456 (2009)
2. Chen, X., Deng, X., Teng, S.: Settling the complexity of computing two-player Nash equilibria. Journal of the ACM 56(3), 1–57 (2009)
3. Crescenzi, P., Papadimitriou, C.: Reversible simulation of space-bounded computations. Theoretical Computer Science 143(1), 159–165 (1995)
4. Daskalakis, C., Goldberg, P.W., Papadimitriou, C.H.: The complexity of computing a Nash equilibrium. SIAM Journal on Computing 39(1), 195–259 (2009)
5. Goldberg, P., Papadimitriou, C., Savani, R.: The Complexity of the Homotopy Method, Equilibrium Selection, and Lemke-Howson Solutions. In: Proceedings of 52nd FOCS. pp. 67–76 (2011)
6. Harsanyi, J., Selten, R.: A General Theory of Equilibrium Selection in Games. MIT Press (1988)
7. Krentel, M.: The complexity of optimization problems. Journal of Comput. System Sci 36(3), 490–509 (1988)
8. Lemke, C., Howson Jr., J.: Equilibrium points of bimatrix games. SIAM J. Appl. Math 12(2), 413–423 (1964)
9. Papadimitriou, C.: On the complexity of the parity argument and other inefficient proofs of existence. J. Comput. Syst. Sci. 48(3), 498–532 (1994)